Chapter 3
Dynamic Programming

Part 2

Algorithm Theory
WS 2012/13

Fabian Kuhn

Dynamic Programming

„Memoization“ for increasing the efficiency of a recursive solution:

- Only the *first time* a sub-problem is encountered, its solution is computed and then stored in a table. Each subsequent time that the subproblem is encountered, the value stored in the table is simply looked up and returned
  (without repeated computation!).

- *Computing the solution*: For each sub-problem, store how the value is obtained (according to which recursive rule).

Dynamic Programming

Dynamic programming / memoization can be applied if

- Optimal solution contains optimal solutions to sub-problems (recursive structure)
- Number of sub-problems that need to be considered is small

polynomial

String Matching Problems

Edit distance:

- For two given strings A and B, efficiently compute the **edit distance $D(A,B)$** (# edit operations to transform A into B) as well as a minimum sequence of edit operations that transform A into B.

A B

- **Example:** mathematician → multiplication:

m u t i p l a t i o ~~i a~~ n

l ic

String Matching Problems

Edit distance $\boldsymbol{D(A,B)}$ (between strings A and B):

```
m a – t h e m - - a t i c i a n
m u l t i p l i c a t i o - - n
```

Approximate string matching:

For a given text *T*, a pattern *P* and a distance *d*, find all

substrings P' of T with $D(P,P') \leq d$.

Sequence alignment:

Find optimal alignments of DNA / RNA / ... sequences.

```
G A G C A - C T T G G A T T C T C G G
- - - C A C G T G G - A - A C T - - -
```

Edit Distance

Given: Two strings $A = a_1 a_2 \dots a_m$ and $B = b_1 b_2 \dots b_n$

Goal: Determine the minimum number $D(A, B)$ of edit operations required to transform A into B

Edit operations:

a) **Replace** a character from string A by a character from B

b) **Delete** a character from string A

c) **Insert** a character from string B into A

```
m a - t h e m - - a t i c i a n
m u l t i p l i c a t i o - - n
```

a) c) b)

Edit Distance – Cost Model

- Cost for **replacing** character a by b: $\boldsymbol{c(a,b) \geq 0}$

- Capture insert, delete by allowing $a = \varepsilon$ or $b = \varepsilon$:
  - Cost for **deleting** character a: $\boldsymbol{c(a,\varepsilon)}$
  - Cost for **inserting** character b: $\boldsymbol{c(\varepsilon,b)}$

- **Triangle inequality**:

$$c(a,c) \leq c(a,b) + c(b,c)$$

→ each character is changed at most once!

- **Unit cost model**: $c(a,b) = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{if } a = b \end{cases}$

Recursive Structure

- Optimal "alignment" of strings (unit cost model)

  `bbcadfagikccm` and `abbagflrgikacc`:

A	-	b	b	c	a	g	f	a	–	g	i	k	-	c	c	m
B	a	b	b	–	a	d	f	l	r	g	i	k	a	c	c	–
	ins.			del		repl.		repl.	ins				ins			del

- Consists of optimal "alignments" of sub-strings, e.g.:

```
-bbcagfa         –gik-ccm
           and
abb-adfl         rgikacc-
```

$A_{i,j} = a_i \dots a_j$

- Edit distance between $A_{1,m} = a_1 \dots a_m$ and $B_{1,n} = b_1 \dots b_n$:

$$D(A,B) = \min_{k,\ell} \{ D(A_{1,k}, B_{1,\ell}) + D(A_{k+1,m}, B_{\ell+1,n}) \}$$

$k < m,\ \ell < n$

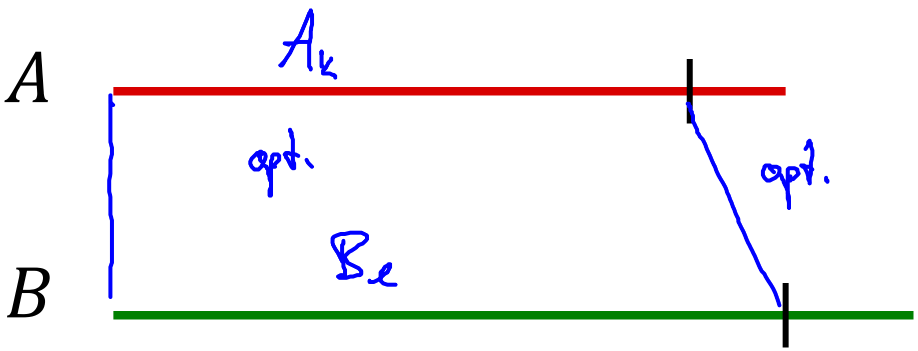
Computation of the Edit Distance

$A_{1,k}$

Let $A_k := a_1 \ldots a_k$, $B_\ell := b_1 \ldots b_\ell$, and

$$D_{k,\ell} := D(A_k, B_\ell)$$

$D_{m,n} = D(A, B)$



Computation of the Edit Distance

Three ways of ending an "alignment" between A_k and B_ℓ:

1. a_k is replaced by b_ℓ:

   $\underline{D_{k,\ell}} = \underline{D_{k-1,\ell-1}} + \underline{c(a_k, b_\ell)}$

   $$\begin{array}{ll|l} a_1 & \cdots\cdots & a_{k-1} & a_k \\ b_1 & \cdots\cdots & b_{\ell-1} & b_\ell \end{array}$$

2. a_k is deleted:

   $\underline{D_{k,\ell}} = \underline{D_{k-1,\ell}} + \underline{c(a_k, \varepsilon)}$

   $$\begin{array}{ll|l} a_1 & \cdots\cdots & a_{k-1} & a_k \\ b_1 & \cdots\cdots & b_\ell & - \end{array}$$

3. b_ℓ is inserted:

   $\underline{D_{k,\ell}} = \underline{D_{k,\ell-1}} + \underline{c(\varepsilon, b_\ell)}$

   $$\begin{array}{ll|l} a_1 & \cdots\cdots & a_k & - \\ b_1 & \cdots\cdots & b_{\ell-1} & b_\ell \end{array}$$

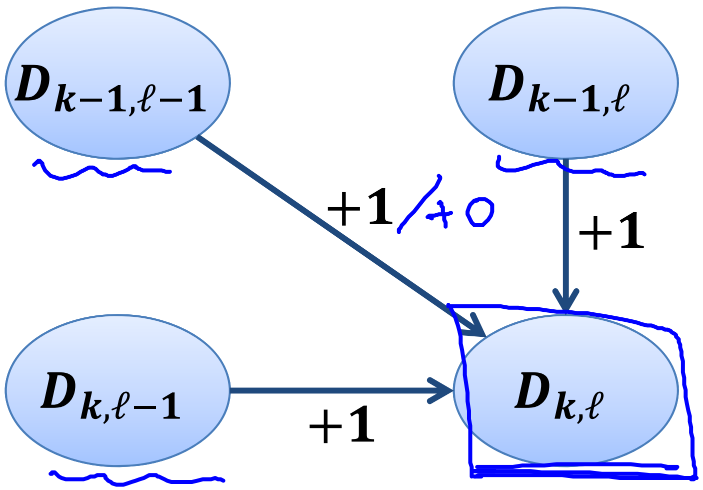
Computing the Edit Distance

- Recurrence relation (for $k, \ell \geq 1$)

0 if $a_k = b_\ell$

$$D_{k,\ell} = \min \begin{Bmatrix} D_{k-1,\ell-1} & + c(a_k, b_\ell) \\ D_{k-1,\ell} & + c(a_k, \varepsilon) \\ D_{k,\ell-1} & + c(\varepsilon, b_\ell) \end{Bmatrix} = \underbrace{\min \begin{Bmatrix} D_{k-1,\ell-1} & + 1 \\ D_{k-1,\ell} & + 1 \\ D_{k,\ell-1} & + 1 \end{Bmatrix}}_{\textbf{unit cost model}}$$

- Need to compute $D_{i,j}$ for all $0 \leq i \leq k$, $0 \leq j \leq \ell$:



Recurrence Relation for the Edit Distance

Base cases:

unit cost

$$D_{0,0} = D(\varepsilon,\varepsilon) = 0$$

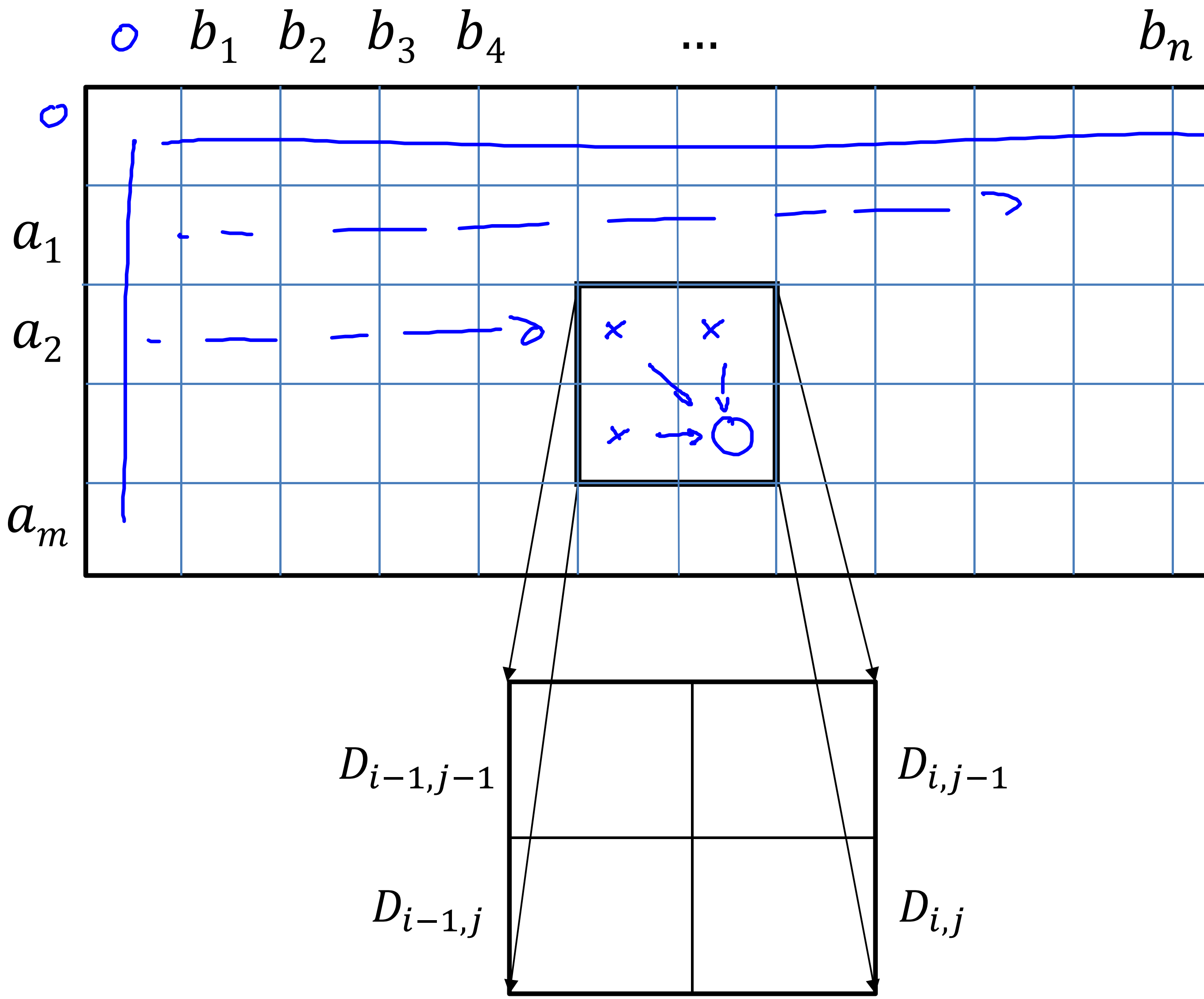
$$D_{0,j} = D(\varepsilon, B_j) = D_{0,j-1} + c(\varepsilon, b_j) = j$$

$$D_{i,0} = D(A_i, \varepsilon) = D_{i-1,0} + c(a_i, \varepsilon) = i$$

Recurrence relation:

$$\boxed{D_{i,j} = \min \begin{Bmatrix} D_{k-1,\ell-1} + c(a_k, b_\ell) \\ D_{k-1,\ell} + c(a_k, \varepsilon) \\ D_{k,\ell-1} + c(\varepsilon, b_\ell) \end{Bmatrix}}$$

Order of solving the subproblems



Algorithm for Computing the Edit Distance

Algorithm *Edit-Distance*

Input: 2 strings $A = a_1 \dots a_m$ and $B = b_1 \dots b_n$

Output: matrix $D = \left(D_{ij}\right)$

$D[0,0] := 0;$

for $i := 1$ **to** m **do** $D[i,0] := i;$

for $j := 1$ **to** n **do** $D[0,j] := j;$

initialize

for $i := 1$ **to** m **do** ← rows

  for $j := 1$ **to** n **do** ← columns

$$D[i,j] := \min \begin{Bmatrix} D[i-1,j] & +1 \\ D[i,j-1] & +1 \\ D[i-1,j-1] + c(a_i,b_j) & \end{Bmatrix};$$

Example

unit cost

		a	b	c	c	a
	0	1	2	3	4	5
b	1	1	1	2	3	4
a	2	1	2	2	3	3
b	3	2	1	2	3	4
d	4	3	2	2	3	4
a	5	4	3	3	3	3

$D_{5,5}$

- a b c c a
b a b d - a

Computing the Edit Operations

Algorithm *Edit-Operations*(i, j)
Input: matrix D (already computed)
Output: list of edit operations

if $i = 0$ **and** $j = 0$ **then return** empty list

if $i \neq 0$ **and** $D[i, j] = D[i-1, j] + 1$ **then**
    return *Edit-Operations*$(i-1, j) \circ$ „delete a_i“

else if $j \neq 0$ **and** $D[i, j] = D[i, j-1] + 1$ **then**
    return *Edit-Operations*$(i, j-1) \circ$ „insert b_j“

else // $D[i, j] = D[i-1, j-1] + c(a_i, b_j)$
    if $a_i = b_i$ **then return** *Edit-Operations*$(i-1, j-1)$
    else return *Edit-Operations*$(i-1, j-1) \circ$ „replace a_i by b_j“

Initial call: *Edit-Operations*(*m*,*n*)

Edit Operations

	a	b	c	c	a
0	1	2	3	4	5
b	1	1	1	2	3	4
a	2	1	2	2	3	3
b	3	2	1	2	3	4
d	4	3	2	2	3	4
a	5	4	3	3	3	3

Edit Distance: Summary

- Edit distance between two strings of length m and n can be computed in $\boxed{O(mn)}$ time.

- Obtain the edit operations:
  - for each cell, store which rule(s) apply to fill the cell
  - track path backwards from cell (m, n) $O(m+n)$
  - can also be used to get all optimal "alignments"

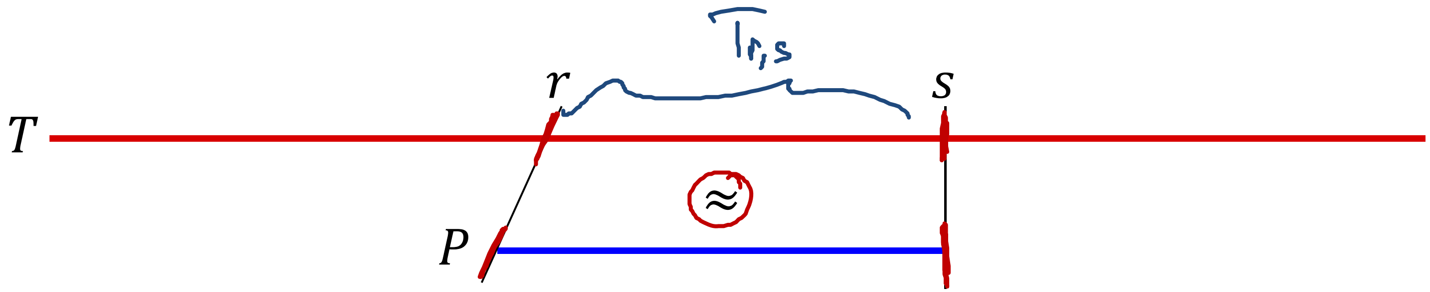
- Unit cost model:
  - interesting special case
  - each edit operation costs 1

Approximate String Matching

Given: strings $T = t_1 t_2 \dots t_n$ (text) and $P = p_1 p_2 \dots p_m$ (pattern).

Goal: Find an interval $[r, s]$, $1 \leq r \leq s \leq n$ such that the sub-string $T_{r,s} := t_r \dots t_s$ is the one with highest similarity to the pattern P:

$$\arg\min_{1 \leq r \leq s \leq n} \boxed{D(T_{r,s}, P)}$$



Approximate String Matching

Naive Solution:

for all $1 \leq r \leq s \leq n$ **do**

compute $D(T_{r,s}, P)$

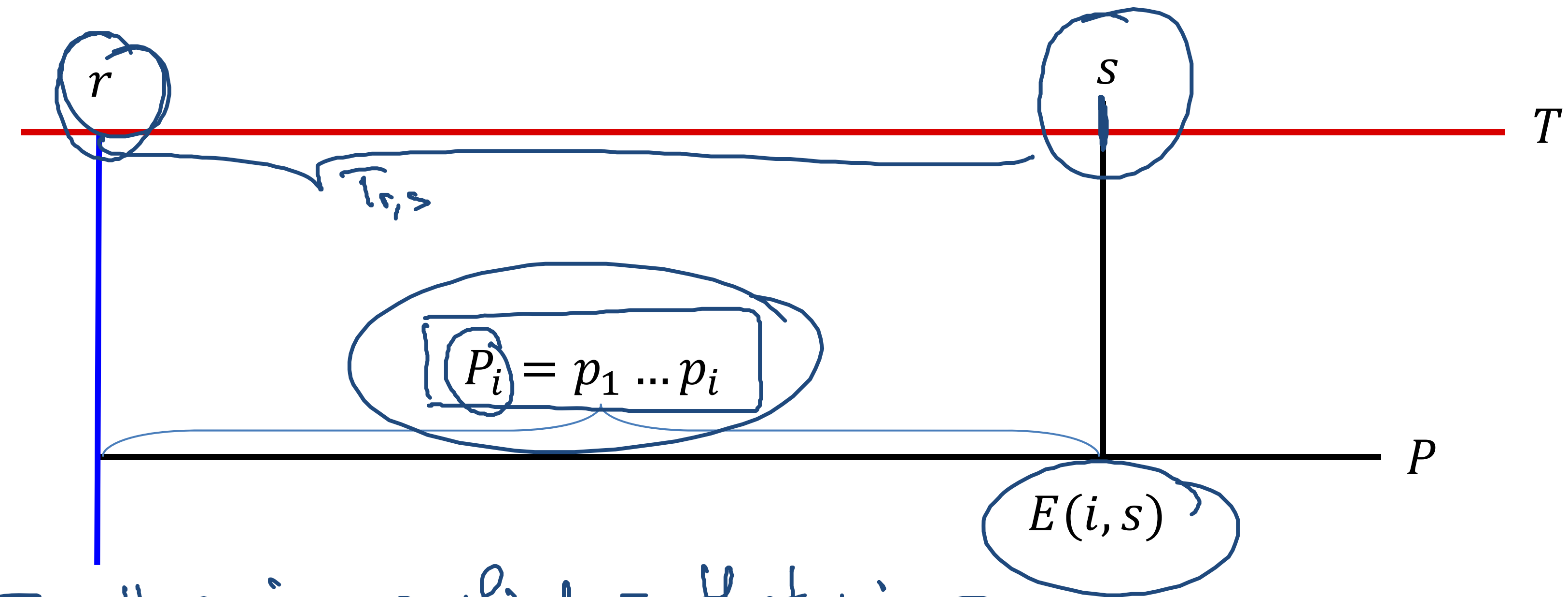
choose the minimum

$O(n^2 \cdot n \cdot m) = O(n^3 m)$

Approximate String Matching

A related problem:

- For each position s in the text and each position i in the pattern compute the minimum edit distance $E(i,s)$ between $P_i = p_1 \dots p_i$ and any substring $T_{r,s}$ of T that ends at position s.



For all s, i $\rightarrow$ find r that min. $\underbrace{D(T_{r,s}, P_i)}_{E(i,s)}$

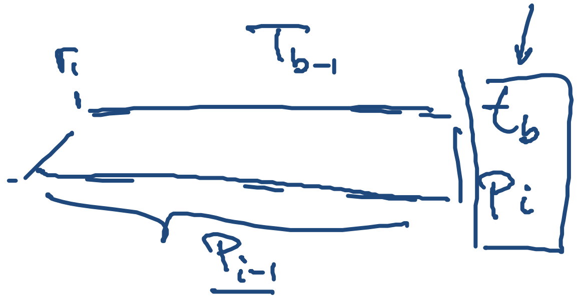
Approximate String Matching

S from previous slide

Three ways of ending optimal alignment between T_b and P_i:

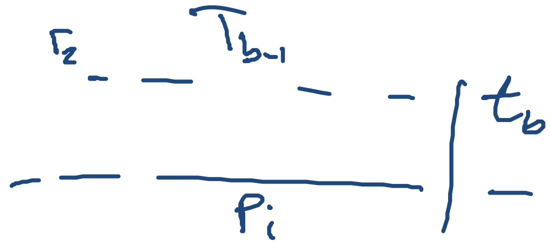
1. t_b is replaced by p_i:

   $$E_{b,i} = \underline{E_{b-1,i-1}} + \underline{c(t_b, p_i)}$$



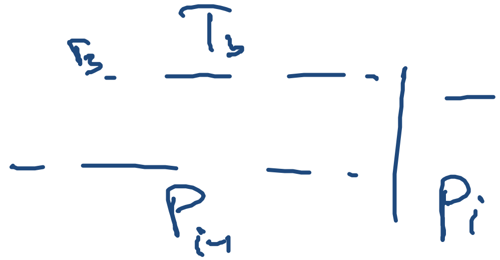
2. t_b is deleted:

   $$E_{b,i} = E_{b-1,i} + c(t_b, \varepsilon)$$



3. p_i is inserted:

   $$E_{b,i} = E_{b,i-1} + c(\varepsilon, p_i)$$



Approximate String Matching

Recurrence relation (unit cost model):

0 if $t_b = p_i$

$$E_{b,i} = \min \begin{Bmatrix} E_{b-1,i-1} & +1 \\ E_{b-1,i} & +1 \\ E_{b,i-1} & +1 \end{Bmatrix}$$

Base cases:

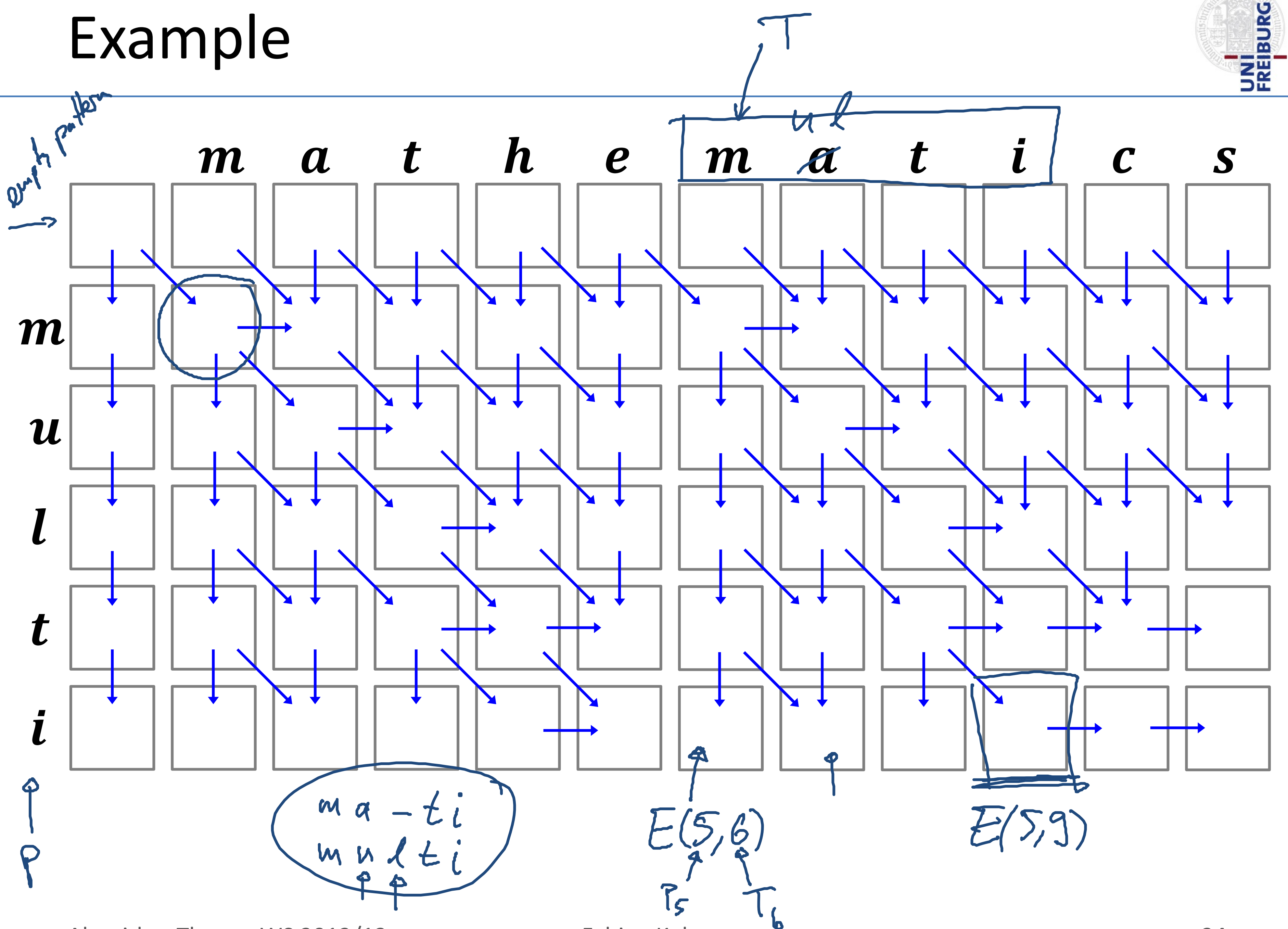
$T = \varepsilon$, P_i

$$E_{0,0} = 0$$
$$E_{0,i} = i$$
$$E_{i,0} = 0$$

$T_i = t_1 \ldots t_i$

P_0

Example


T
ul
empty pattern
m a t h e m a t i c s
m
u
l
t
i
P
ma-ti
multi
E(5,6)
P5
T6
E(5,9)

Approximate String Matching

- Optimal matching consists of optimal sub-matchings

$m << n$

- Optimal matching can be computed in $O(mn)$ time

- Get matching(s):
  - Start from minimum entry/entries in bottom row
  - Follow path(s) to top row

- Algorithm to compute $E(b, i)$ identical to edit distance algorithm, except for the initialization of $E(b, 0)$

Related Problems from Bioinformatics

Sequence Alignment:

Find optimal alignment of two given DNA, RNA, or amino acid sequences.

```
G A – C G G A T T A G
G A T C G G A A T - G
```

Global vs. Local Alignment:

- *Global alignment*: find optimal alignment of 2 sequences
- *Local alignment*: find optimal alignment of sequence 1 (pattern) with sub-sequence of sequence 2 (text)